Alexander Garcia June 5, 2017

Assignment-2

1. (a) Consider the functions $f(x) = x^3 - 2,\ g(x) = e^x - 5sin(x^3) - 3cos(x)$.

The initial step in the bisection process is to choose a range of x in which $f(x) = 0$. As the range is given to be $(0, 2)$, the bisection algorithm proceeds by guessing the midpoint of the range (let's call this $x_0 = \frac{b_0 - a_0}{2}$) to be a zero of the function. If $|f(x_0)| \leq 0 + tol$, then the zero is said to have been found. If $|f(x_0)| > 0 + tol$, then either a_0 or b_0 is replaced by x_0, depending on where $f(x)$ changes sign. The process is then repeated until $f(x_n) < 0 + tol$.

At each stage in the algorithm, the maximum possible error of $f(x_n)$ is $e_n = \frac{b_n - a_n}{2}$, and the general error is $e_n \leq \frac{b_n - a_n}{2}$. Because $b_n - a_n = \frac{b_{n-1} - a_{n-1}}{2}$, we can express the current error as

$$e_n \leq \frac{1}{2} e_{n-1}$$

In this case we are taking 34 iterations until e_n is reasonably low, so $e_{34} = \frac{1}{2} e_{33}$. It is clear that this expression will cascade down to e_0, the error of the initial guess. The final error e_{34} can therefore be generalized to

$$e_0 * \prod_{i=1}^{34} \frac{1}{2} = 1 * \prod_{i=1}^{34} \frac{1}{2} = \frac{1}{2^{34}}$$

This result shows that the errors of the bisection method do not rely at all upon the function whose zeroes are being calculated, but rather solely upon the number of iterations carried out by the process. The number of iterations to calculate $g(x) = 0$ to sufficent accuracy is exactly the same number of iterations needed to calculate $f(x) = 0$.

(b) The convergence rate of a root finding algorithm is the limit of the ratio of the error in iteration n to the error in iteration $n - 1$ as $n \to \infty$.

$$conv = \lim_{n \to \infty} \frac{e_n}{e_{n-1}}$$

i. This algorithm is linearlly convergent, or has a constant convergance rate, as the ratio between every adjacent error is $\frac{1}{2}$.

ii. This algorithm is quadratically convergent. It is clear that the ratio of e_n to e_{n-1} is cut in half after every iteration, making the rate of convergance not a constant factor, but a result of which iteration is currently taking place.

$$(\frac{e_n}{e_{n-1}} = \frac{1}{2^n})|_{n=1}^{\infty}$$

(c) For a "good" starting guess when undergoing Newton approximation, the rate of convergence is quadratic.

After choosing a good starting guess, the next approximations for $f(x) = 0$ are determined by $x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$, where x_n approaches the true zero, $x = \alpha$. At each stage, the error is $e_n = |x_n - \alpha|$, and the rate of convergance would be $\frac{e_n}{e_{n+1}}$.

We let the function $f(x)$ be approximated by the first order Taylor polynomial about x_n.

$$f(x) = f(x_n) + (x - x_n)f'(x_n) + \frac{1}{2}(x - x_n)^2 f''(c_n)$$

where c_n is a constant close to x_n used to estimate the remainder. If we evaluate the approximation at $x = \alpha$, we would ideally get zero, as this is the true root of the function.

$$0 = f(x_n) + (\alpha - x_n)f'(x_n) + \frac{1}{2}(\alpha - x_n)^2 f''(c_n)$$

If we then divide by $f'(x_n)$, we see that the expression for x_{n+1} appears.

$$0 = \frac{f(x_n)}{f'(x_n)} + (\alpha - x_n) + \frac{(\alpha - x_n)^2}{2}\frac{f''(c_n)}{f'(x_n)}$$

$$x_n - \frac{f(x_n)}{f'(x_n)} - \alpha = \frac{(\alpha - x_n)^2}{2}\frac{f''(c_n)}{f'(x_n)}$$

$$x_{n+1} - \alpha = \frac{1}{2}(\alpha - x_n)^2\frac{f''(c_n)}{f'(x_n)}$$

We know that by definition, $x_n - \alpha = e_n$, and that $\lim_{n\to\infty} x_n = \alpha$, so this can be rewritten as

$$\frac{e_{n+1}}{{e_n}^2} = \frac{1}{2}\frac{f''(\alpha)}{f'(\alpha)}$$

Because the right hand side is constant, and the ratio of errors is of the form $\frac{R}{R^2} = c$, we can say that the sequence is quadratically convergant. Or, if $c = 1$, we can say that the sequence is linearlly convergant.

If a poor starting guess is chosen, Newton's method will not converge at all towards a root of the function. A "bad" starting guess would be in the region [a,b] about the root such that

- f, f', f'' does not exist in [a,b]
- $f' = 0$ in [a,b]
- f'' changes sign in [a,b]

Therefore, **iv** is correct.

(d) True. If an initial guess for Newton's root finding method is to be considered "good", it must follow some criteron.

- f, f', f'' must exist exist
- Choose a, b such that $f(a) * f(b) < 0$
- $f' \neq 0$ in $[a, b]$
- f'' does not change sign in $[a, b]$
- $|\frac{f(a)}{f'(a)}|$ and $|\frac{f(b)}{f'(b)}| < |b - a|$

If all of these hold true, the there will be a unique root in $[a, b]$, and any starting guess in $[a, b]$ will converge to the root. Therefore, if x_0 leads to convergance, then any guess between x^* and x_0 will also lead to convergance.

(e) One of the main advantages of Newton's method over the bisection method is the speed at which it can approximate roots. The quadratic convergance rate of Newton's method means that given a good starting guess, it can approximate a root in a fraction of the time of the bisection method.

However, unlike the bisection method, it requires a good starting guess. If the initial zero chosen does not follow some precise specifications (see part(d)), then the method will not converge. Despite its much longer runtime, the bisection method will always work, given that α lies in the initial range.

Also, Newton's method can be somewhat more difficult to compute, as the f' is needed to compute the root of f. The bisection method relies solely on f.

(f) One of the main advantages of the Secant method over Newton's method is the fact that the secant method does not require $f'(x)$ to approximate the root. However, it requires two starting guesses, and does not converge at the rate of Newton's method (Newton's: quadratic convergance, secant: "superlinear" convergance).

(g) $f(x) = 2 - x^2$
$f'(x) = -2x$
We choose $x_0 = 2$ for a starting guess.

The rule for Newton's method is $x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$

$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)} = \frac{3}{2}$

$x_2 = x_1 - \frac{f(x_1)}{f'(x_1)} = \frac{17}{12}$

$x_3 = x_2 - \frac{f(x_2)}{f'(x_2)} = \frac{577}{408}$

$x_4 = x_3 - \frac{f(x_3)}{f'(x_3)} = \frac{665857}{470832}$

$x_5 = x_4 - \frac{f(x_4)}{f'(x_4)} = \frac{886731088897}{627013566048}$

At this point, if we define the tolerance as 10^{-12}, we must see if we are close enough to the root. Because technically we do not know the true answer $\sqrt{2}$, we see if the last two roots are sufficiently close together.

$x_5 - x_4 = \frac{665857}{470832} - \frac{886731088897}{627013566048} = \frac{1}{627013566048} \approx 1.6 * 10^{-12}$

The result is within the defined tolerance, and the algorithm is complete, with the calculated root being

$$x_5 = \frac{886731088897}{627013566048} \approx 1.41421356237468$$

Compared to the actual value of the root, $\sqrt{2}$, the relative error is $\frac{x_5 - \sqrt{2}}{\sqrt{2}} \approx 1.13 * 10^{-12}$

(h) The bisection method will not converge to a real number answer. The rule of the bisection method is that you begin by defining a domain in which the root lies, which is done by checking where $f(x)$ changes sign. In the case of $f(x) = \frac{1}{x}$, the function only changes sign if the lower bound ($x = a$) is less than zero, and the upper bound ($x = b$) is greater than zero.

As this is the only possible point the zero could be, the bisection algorithm will keep approaching the midpoint of [a,b], which will approach zero. But as we know, $\frac{1}{0+\delta}$ becomes increasingly large as $\delta \to 0$.

Therefore, the bisection method will not converge to a real number answer, which is good, since there are no real number answers to $\frac{1}{x} = 0$.

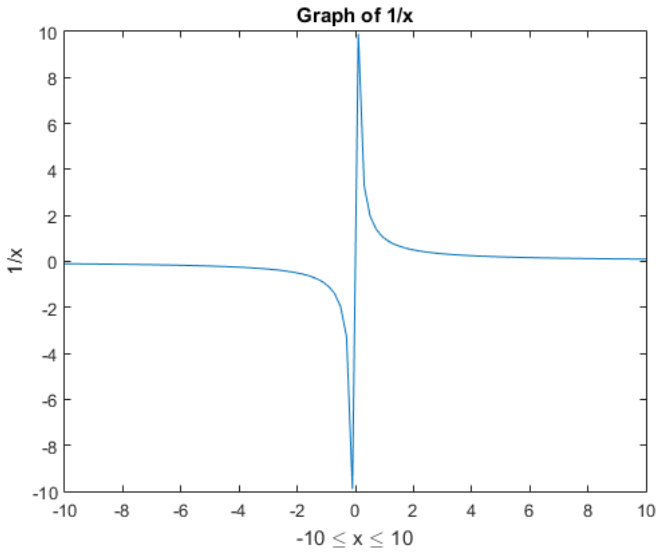

Figure 1: Graph of $\frac{1}{x}$ produced by MATLAB

(i) i. $f(x) = x^2 sinx^2$

$f(x) = (x - r)^m * F(x)$

The exponent on the $(x - r)$ term of the function is, by definition, the multiplicity of the root r. In this case the multiplicity of root $r = 0$ is 2.

ii. Assume we are using Newton's method for root finding. The next approximated zero is defined as $x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$.
If $f'(x) >> 1$ as the approximation approaches the root, then we should use the $f(x_n)$ as the stopping criteria, since if $f'(x_n)$ grows too large, we lose the significance of the result ($\frac{f(x_n)}{f'(x_n)} \approx 0$).
If $f'(x_n) << 1$, we should approximate the zero as $x_{n+1} - x_n$, since a large $f'(x_n) \rightarrow \frac{f(x_n)}{f'(x_n)} \approx \infty$, which again removes significance from the result.

In this case, $f'(x) = 2xsinx^2 + x^2 cosx^2$. As $x \rightarrow 0$, the function will grow increasingly small. Therefore, we should accept the result when $|x_{n+1} - x_n| < tol$.

2. Holmes 2.4 (B)

(a) See Figure 2

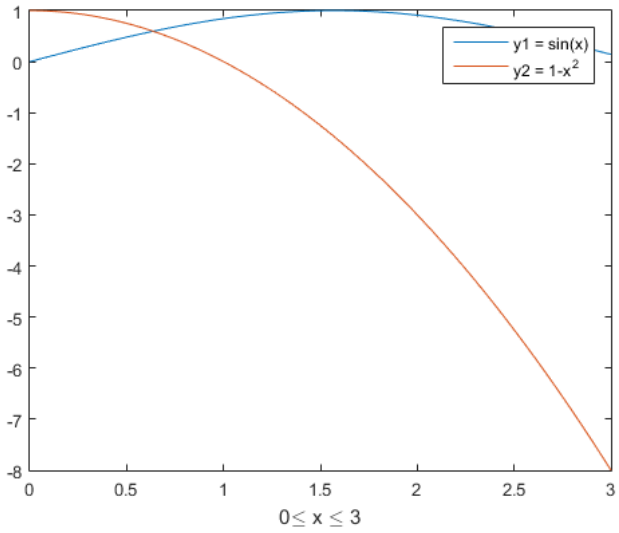

Figure 2: Graph of $sinx$ and $1 - x^2$

(b) We can treat the equation $sinx = 1 - x^2$ as $sinx + x^2 - 1 = 0$, where we are solving for the roots of the function. It is clear that the intersection of the two points occurrs somewhere between $x = 0.5$ and $x = 1$, so we choose $a = 0.5$ and $b = 1$. We can see that the root lies within this range, since $f(0.5) \approx -0.27057\ldots$, and $f(1) \approx 0.84147\ldots$, meaning that $f(a) * f(b) < 0$. We have shown that the function changes sign in this interval, and according to the Intermediate Value Theorem, if a function is continuous over an interval, then the function must take every value between $(f(a), f(b)$. In this case, it means that if a function is negative on one end of the interval and positive on the other, the zero must lie somewhere in the interval.

The initial guess, c_0 would simply be the midpoint of a and b.

$$c_0 = \frac{b+a}{2} = \frac{1.5}{2} = 0.75$$

We must then test to see whether a is replaced by c_0, or if b is replaced by c_0.

$$f(c_0) \approx 0.24414\ldots$$

Because the root must lie in the domain where $f(x)$ changes signs, we must replace a with c_0, as $f(a) > 0$.
We now repeat the process to find c_1

$$c_1 = \frac{b+c_0}{2} = \frac{1.25}{2} = 0.625$$

(c) The general form for each iteration of Newton's method is

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

In this case, $f(x) = sinx + x^2 - 1$, and $f'(x) = cosx + 2x$. The only element left is x_0. This must be chosen carefully, as a poor initial guess will not lead to convergance. A good guess will follow the rules outlined in Question 1 part (d). As a quick check,

- $f(x) = sinx + x^2 - 1,\ f^{'}(x) = cosx + 2x,\ f^{''}(x) = 2 - sinx$
- If $a = 0$ and $b = 1$, we can see that $f(a) = -1,\ f(b) = sin(1)$, so $f(a) * f(b) < 0$. This means there is a root in the range $[a, b]$.
- As shown in Figure 3, the first derivative of $f(x)$ is nonzero for all $0 \leq x \leq 1$.
- As shown in Figure 3, the second derivative of $f(x)$ is positive for all $0 \leq x \leq 1$.
- $|\frac{f(a)}{f'(a)}| = \frac{1}{1} \leq 1 - 0$
  $|\frac{f(b)}{f'(b)}| = \frac{sin(1)}{cos(1)+2} \approx 0.33125 \leq 1 - 0$

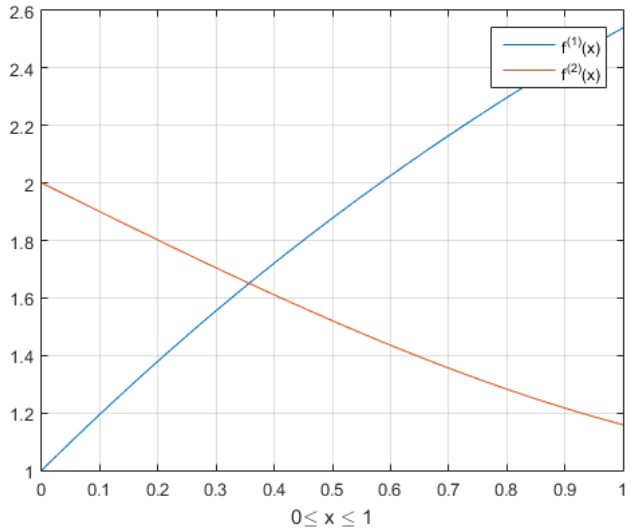

Figure 3: Graph of $f^{'}(x)$ and $f^{''}(x)$

We can now proceed with a starting guess anywhere in the range $0 \leq x_0 \leq 1$. As it is the easiest, choose $x_0 = 0$.

$$x_1 = 0 - \frac{f(0)}{f'(0)} = -\frac{-1}{1} = 1$$

(d) The general form for the secant method is

$$x_{n+2} = x_n - \frac{f(x_n)(x_{n+1} - x_n)}{f(x_{n+1}) - f(x_n)} = x_n - \frac{f(x_n)}{m_n}$$

A good choice for starting points x_0, x_1 would be points close to the true root of $f(x)$ which will lead the algorithm to convergance.

More formally, let us let $f(\overline{x}) = 0$. If $f(x)$ is twice differentiable in $a < \overline{x} < b$ and $f'(x) \neq 0$ in (a, b), then any x_0, x_1 chosen close to $\overline{x}$ will converge to $\overline{x}$ through the secant method.

As the root of this function is $0 < \overline{x} < 1$, we can choose $x_0 = 1, x_1 = 0$.

$$x_2 = 1 - \frac{f(1)(0-1)}{f(0) - f(1)} = 1 - \frac{-sin(1)}{-1 - sin(1)} \approx 0.54304$$

(e) In order to compute the exact solution to the equation, I used a MATLAB script defining Newton's Method. I chose this method as it converges the fastest out of the three algorithms, and only requires a single starting guess. For error, I used 10^{-15} as the stopping criteria. All credit for the MATLAB script goes to professor Kapila.

$$\overline{x} = 0.6367$$

Definition of $f(x)$ and $f'(x)$

```
% Definitions of f(x) and f'(x)
y = sin9x)+x.^2-1;
yp = cos(x)+2*x;
```

Script used to calculate $\overline{x}$

```
function r = Newton(fun,x0,tol)
%
% Input: fun = handle of function, defined in an m-file that
% returns f(x) and f'(x)
% x0 = initial guess
% tol = tolerance for stopping
%
% Output: r = the root of f
x = x0; % initial guess
k = 0; % initialize iteration counter
```

```
maxit=20; % max iterations allowed
while k<= maxit
    k=k+1;
    [f,fp] = fun(x);
    dx = f/fp;
    x = x-dx;
    err = abs(dx);
    fprintf('\n k = %i Solution = %20.15e Error = %15.10e \n',k,x,err);
if (err < tol), r=x; return; end
end
warning(sprintf('root not found within tolerance after %d iterations\n',k));
%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
>> r=Newton(@ff,1,1e-15)

 k = 1 Solution = 6.687516352427485e-01 Error = 3.3124836476e-01

 k = 2 Solution = 6.370680330525878e-01 Error = 3.1683602190e-02

 k = 3 Solution = 6.367326888384163e-01 Error = 3.3534421417e-04

 k = 4 Solution = 6.367326508052825e-01 Error = 3.8033133804e-08

 k = 5 Solution = 6.367326508052821e-01 Error = 4.2752105449e-16

r =

    0.6367
```

Figure 4: Printed output of script in Figure 5

3. Consider the function $f(x) = e^{sin^3 x} + x^6 - 2x^4 - x^3 - 1, -2 \leq x \leq 2$

   (a) See Figure 7

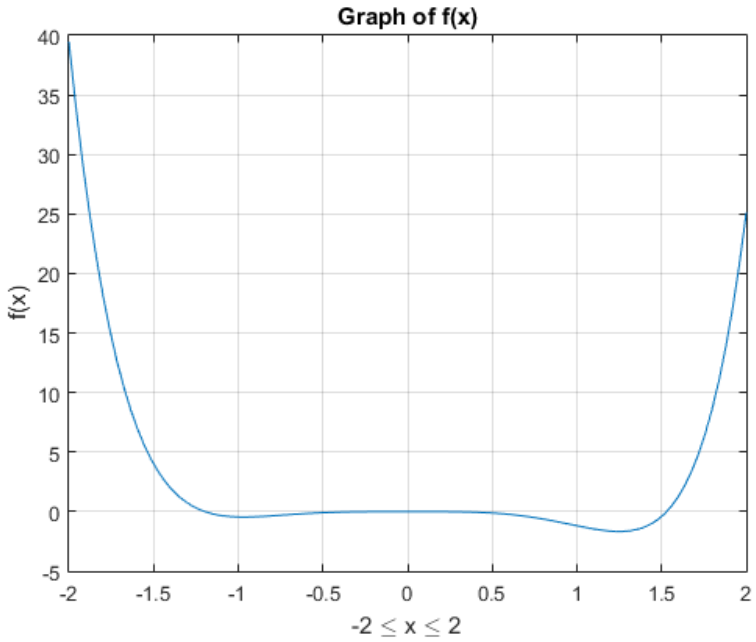

Figure 5: Graph of $f(x)$

   (b) For the calculation of roots $r_n,\ n = 1, 2, 3$, the tolerance was defined by $x_{n+1} - x_n$. This is due to the fact that the derivative is very small about the roots, so there would be some loss of significance when calculating the roots due to the size of the resulting fraction.

Function definitions of $f(x)$ and $f'(x)$

```
% generate y = f, yp=f' from given function input
function [y,yp] = f(x)
y = exp(sin(x).^3)+x.^6-2*x.^4-x.^3-1;
yp = 3*exp(sin(x).^3).*(sin(x)).^2.*cos(x)+6*x.^5-8*x.^3-3*x.^2;
```

Definition of Newton's method for root finding

```
function [x,n] = myNewton(fun, x0, tol)
% input:
% x0 = initial guess
% tol = tolerance allowed for error
%  - tol calculated by (x_n+1)-(x_n), as f'(x) is small close to f(x)=0
% fun = function whose zeroes are being calculated
%
```

```
% output:
% x = calculated root of f(x)
% n = number of iterations needed to calculate root
x = x0; % initial guess
n = 0;
% initialize other vars to zero
err = 0;
dx = 0;

while n <= 20 % set iteration limit to 20, just in case
    n = n+1;
    [y,yp] = fun(x); % assign functions defined in f.m
    dx = y/yp;
    xn = x-dx; % calculate next root
    err = abs(xn-x); % calculate deviation between each root
    fprintf('\n n = %i Solution = %12.7e    Error = %7.7e \n',n,x,err);
    x = xn;
    if err<=tol, return; end
end
warning(sprintf('root not found within tolerance after %d iterations\n',n));
```

`callNewton.m` script to calculate zeroes of $f(x)$

```
% script to call myNewton function
% supplies initial guess 'x0' and error tolerance 'tol'

% global tolerance
tol = 1e-15;

% first guess
fprintf('First root guess');
g1 = -2;
[r1,n1] = myNewton(@f, g1, tol);

% second guess
fprintf('\nSecond root guess');
g2 = -1;
[r2,n2] = myNewton(@f, g2, tol);

% third guess
fprintf('\nThird root guess');
g3 = 2;
[r3,n3] = myNewton(@f, g3, tol);
```

Output of `callNewton` script

```
callNewton
First root guess
 n = 1 Solution = -2.0000000e+00    Error = 2.8096256e-01
 n = 2 Solution = -1.7190374e+00    Error = 2.1895787e-01
 n = 3 Solution = -1.5000796e+00    Error = 1.5898695e-01
 n = 4 Solution = -1.3410926e+00    Error = 9.7511340e-02
 n = 5 Solution = -1.2435813e+00    Error = 3.9581864e-02
 n = 6 Solution = -1.2039994e+00    Error = 6.2324884e-03
 n = 7 Solution = -1.1977669e+00    Error = 1.4313208e-04
 n = 8 Solution = -1.1976238e+00    Error = 7.4202301e-08
 n = 9 Solution = -1.1976237e+00    Error = 1.9984014e-14
 n = 10 Solution = -1.1976237e+00    Error = 2.2204460e-16

 Second root guess
  n = 1 Solution = -5.0000000e-01    Error = 1.4676065e-01
  n = 2 Solution = -3.5323935e-01    Error = 9.5631426e-02
  n = 3 Solution = -2.5760793e-01    Error = 6.7372488e-02
  n = 4 Solution = -1.9023544e-01    Error = 4.8869777e-02
  n = 5 Solution = -1.4136566e-01    Error = 3.5951159e-02
  n = 6 Solution = -1.0541450e-01    Error = 2.6648319e-02
  n = 7 Solution = -7.8766184e-02    Error = 1.9838342e-02
  n = 8 Solution = -5.8927843e-02    Error = 1.4806905e-02
  n = 9 Solution = -4.4120938e-02    Error = 1.1069277e-02
  n = 10 Solution = -3.3051660e-02    Error = 8.2835933e-03
  n = 11 Solution = -2.4768067e-02    Error = 6.2031156e-03
  n = 12 Solution = -1.8564951e-02    Error = 4.6472589e-03
  n = 13 Solution = -1.3917693e-02    Error = 3.4827171e-03
  n = 14 Solution = -1.0434975e-02    Error = 2.6105578e-03
  n = 15 Solution = -7.8244177e-03    Error = 1.9571084e-03
  n = 16 Solution = -5.8673093e-03    Error = 1.4673852e-03
  n = 17 Solution = -4.3999241e-03    Error = 1.1002921e-03
  n = 18 Solution = -3.2996320e-03    Error = 8.2508182e-04
  n = 19 Solution = -2.4745502e-03    Error = 6.1873463e-04
  n = 20 Solution = -1.8558155e-03    Error = 4.6401069e-04
  n = 21 Solution = -1.3918049e-03    Error = 3.4798549e-04
 root not found within tolerance after 21 iterations

Third root guess
 n = 1 Solution = 2.0000000e+00    Error = 2.2072493e-01
 n = 2 Solution = 1.7792751e+00    Error = 1.4930150e-01
 n = 3 Solution = 1.6299736e+00    Error = 7.7477968e-02
 n = 4 Solution = 1.5524956e+00    Error = 2.0953809e-02
 n = 5 Solution = 1.5315418e+00    Error = 1.4022868e-03
 n = 6 Solution = 1.5301395e+00    Error = 5.9988826e-06
```

```
n = 7 Solution = 1.5301335e+00    Error = 1.0939849e-10
n = 8 Solution = 1.5301335e+00    Error = 2.2204460e-16
```

Therefore, the roots of the function found by this method are

$$x = -1.1976237,\ 1.5301335$$

Because the middle root $f(0) = 0$ is of multiplicity 2, Newton's method was unable to approximate it.

(c) We can define the order of convergance as $lim_{n\to\infty} \frac{ln(e_{n+1})}{ln(e_n)}$

First root	Third root
$\frac{ln(e_2)}{ln(e_1)} = 1.1964044\ldots$	$\frac{ln(e_2)}{ln(e_1)} = 1.2587634\ldots$
$\frac{ln(e_3)}{ln(e_2)} = 0.69036433\ldots$	$\frac{ln(e_3)}{ln(e_2)} = 1.3449250\ldots$
$\vdots$	$\vdots$
$\frac{ln(e_8)}{ln(e_7)} = 1.85460323\ldots$	$\frac{ln(e_6)}{ln(e_5)} = 1.83022469\ldots$
$\frac{ln(e_9)}{ln(e_8)} = 1.92147535\ldots$	$\frac{ln(e_7)}{ln(e_6)} = 1.90753024\ldots$
$\frac{ln(e_{10})}{ln(e_9)} = 1.14265255\ldots$	$\frac{ln(e_8)}{ln(e_7)} = 1.57148656\ldots$

The second root of the function did not converge under Newton's method. The first root was approaching 2, with the last value as an outlier, meaning it converged quadratically. The third root behaved similarly to the first, with values approaching 2 and a final outlier value. This makes the thrid root's convergance quadratic as well.

4. Consider the equation

$$f(\beta_n) = 1 + \frac{1}{cosh\beta_n + cos\beta_n} - \alpha\beta_n(tan\beta_n - tanh\beta_n) = 0$$

(a) The below text shows the output of the `rootfinder.m` script. `rootfinder.m` calls the built in MATLAB routine `fzero`, and supplies it with the function and initial guess. Options are enabled to print out the steps taken by `fzero`. To summarize, the roots calculated for the various α are as follows:

$\alpha = 0.1$	$x = 1.72274$
$\alpha = 1$	$x = 1.24792$
$\alpha = 10$	$x = 0.735782$
$\alpha = 100$	$x = 0.415934$
$\alpha = 1000$	$x = 0.234021$

`rootfinder.m` script used to calculate roots

```
% Script to find natural frequencies of beam oscillation
% Calculate zeroes using 'fzero' routine

fprintf('Searching for lowest root of a=0.1');
a0 = 0.1;                                % alpha value
x0 = 1.75;                               % initial guess
y = fb(a0);                              % define function
fzero(y,x0,optimset('Display','Iter')); % display MATLAB process

fprintf('\nSearching for lowest root of a=1');
a1 = 1;                                  % alpha value
x1 = 1;                                  % initial guess
y = fb(a1);                              % define function
fzero(y,x1,optimset('Display','Iter')); % display MATLAB process

fprintf('\nSearching for lowest root of a=10');
a2 = 10;                                 % alpha value
x2 = 1;                                  % initial guess
y = fb(a2);                              % define function
fzero(y,x2,optimset('Display','Iter')); % display MATLAB process

fprintf('\nSearching for lowest root of a=100');
a3 = 100;                                % alpha value
x3 = 1;                                  % initial guess
y = fb(a3);                              % define function
fzero(y,x3,optimset('Display','Iter')); % display MATLAB process

fprintf('\nSearching for lowest root of a=1000');
a4 = 1000;                               % alpha value
x4 = 0.1;                                % initial guess
y = fb(a4);                              % define function
fzero(y,x4,optimset('Display','Iter')); % display MATLAB process
```

Definition of function $f(x)$

```
% Definition of the function
function y = fb(a)
y = @(x) 1+(1./(cosh(x).*cos(x)))-a.*x.*(tan(x)-tanh(x));
```

Output of `rootfinder.m` script

```
rootfinder
Searching for lowest root of a=0.1
Search for an interval around 1.75 containing a sign change:
 Func-count     a          f(a)             b          f(b)        Procedure
     1          1.75        0.23814         1.75        0.23814    initial interval
     2        1.7005      -0.269524         1.75        0.23814    search

Search for a zero in the interval [1.7005, 1.75]:
 Func-count     x          f(x)             Procedure
     2          1.75        0.23814         initial
     3       1.72678      0.0406233         interpolation
     4       1.72263    -0.00113861         interpolation
     5       1.72274     3.06366e-05        interpolation
     6       1.72274     2.24927e-08        interpolation
     7       1.72274    -1.11022e-15        interpolation
     8       1.72274    -1.11022e-15        interpolation

Zero found in the interval [1.7005, 1.75]

Searching for lowest root of a=1
Search for an interval around 1 containing a sign change:
 Func-count     a          f(a)             b          f(b)        Procedure
     1             1        1.40362           1        1.40362    initial interval
     3      0.971716         1.4792     1.02828        1.31765    search
     5          0.96         1.5078        1.04        1.27861    search
     7      0.943431        1.54578     1.05657         1.2196    search
     9          0.92        1.59495        1.08        1.12775    search
    11      0.886863        1.65637     1.11314       0.978445    search
    13          0.84        1.72932        1.16       0.718817    search
    15      0.773726        1.80981     1.22627       0.214123    search
    17          0.68        1.88928        1.32      -0.998864    search

Search for a zero in the interval [0.68, 1.32]:
 Func-count     x          f(x)             Procedure
    17          1.32      -0.998864         initial
    18          1.32      -0.998864         interpolation
    19       1.21192        0.34084         interpolation
    20       1.23942      0.0876927         interpolation
    21       1.24817    -0.00270616         interpolation
    22       1.24791     7.62143e-05        interpolation
    23       1.24792     6.42978e-08        interpolation
    24       1.24792    -1.33227e-15        interpolation
    25       1.24792    -1.33227e-15        interpolation

Zero found in the interval [0.68, 1.32]
```

```
Searching for lowest root of a=10
Search for an interval around 1 containing a sign change:
 Func-count     a          f(a)             b          f(b)         Procedure
    1           1       -5.75871            1       -5.75871    initial interval
    3    0.971716        -4.7749      1.02828       -6.87791    search
    5        0.96       -4.40266         1.04       -7.38619    search
    7    0.943431       -3.90835      1.05657       -8.15456    search
    9        0.92       -3.26844         1.08       -9.35058    search
   11    0.886863       -2.46916      1.11314       -11.2953    search
   13        0.84       -1.52015         1.16       -14.6777    search
   15    0.773726       -0.47306      1.22627       -21.2553    search
   16        0.68       0.560367      1.22627       -21.2553    search

Search for a zero in the interval [0.68, 1.22627]:
 Func-count     x          f(x)             Procedure
   16        0.68       0.560367         initial
   17    0.694032       0.433024         interpolation
   18    0.740794     -0.0578764         interpolation
   19    0.735281      0.00571623        interpolation
   20    0.735776     6.59543e-05        interpolation
   21    0.735782    -1.25764e-09        interpolation
   22    0.735782     1.77636e-14        interpolation
   23    0.735782    -4.44089e-16        interpolation
   24    0.735782    -4.44089e-16        interpolation

Zero found in the interval [0.68, 1.22627]

Searching for lowest root of a=100
Search for an interval around 1 containing a sign change:
 Func-count     a          f(a)             b          f(b)         Procedure
    1           1       -77.3819            1       -77.3819    initial interval
    3    0.971716       -67.3159      1.02828       -88.8335    search
    5        0.96       -63.5072         1.04       -94.0342    search
    7    0.943431       -58.4496      1.05657       -101.896    search
    9        0.92       -51.9024         1.08       -114.134    search
   11    0.886863       -43.7246      1.11314       -134.032    search
   13        0.84       -34.0148         1.16       -168.642    search
   15    0.773726       -23.3018      1.22627       -235.949    search
   17        0.68       -12.7287         1.32       -397.817    search
   19    0.547452       -4.06133      1.45255       -1087.72    search
   20        0.36        0.88001      1.45255       -1087.72    search

Search for a zero in the interval [0.36, 1.45255]:
 Func-count     x          f(x)             Procedure
   20        0.36        0.88001         initial
   21    0.360883       0.868948         interpolation
   22    0.430207       -0.29059         interpolation
   23    0.412834      0.0592454         interpolation
   24    0.415776     0.00305382         interpolation
   25    0.415934    -2.69158e-06        interpolation
   26    0.415934      1.555e-09         interpolation
```

```
   27         0.415934   -4.44089e-16         interpolation
   28         0.415934   -4.44089e-16         interpolation

Zero found in the interval [0.36, 1.45255]

Searching for lowest root of a=1000
Search for an interval around 0.1 containing a sign change:
 Func-count     a          f(a)             b          f(b)        Procedure
    1             0.1       1.93335           0.1       1.93335   initial interval
    3       0.0971716       1.94058      0.102828       1.92548   search
    5           0.096       1.94339         0.104       1.92203   search
    7       0.0943431        1.9472      0.105657       1.91694   search
    9           0.092       1.95225         0.108       1.90932   search
   11       0.0886863       1.95877      0.111314       1.89767   search
   13           0.084       1.96682         0.116       1.87932   search
   15       0.0773726       1.97611      0.122627       1.84928   search
   17           0.068       1.98575         0.132       1.79764   search
   19       0.0547452       1.99401      0.145255       1.70328   search
   21           0.036       1.99888         0.164        1.5178   search
   23      0.00949033       1.99999       0.19051       1.12187   search
   25          -0.028       1.99959         0.228      0.198105   search
   27      -0.0810193       1.97128      0.281019      -2.16087   search

Search for a zero in the interval [-0.0810193, 0.281019]:
 Func-count     x          f(x)             Procedure
   27      -0.0810193       1.97128         initial
   28       0.0916948       1.95288         interpolation
   29        0.186357       1.19598         bisection
   30        0.233688     0.0113561         bisection
   31        0.234068   -0.00162468         interpolation
   32        0.234021   3.47205e-06         interpolation
   33        0.234021   1.05803e-09         interpolation
   34        0.234021   5.77316e-15         interpolation
   35        0.234021   -4.44089e-15        interpolation

Zero found in the interval [-0.0810193, 0.281019]
```

(b) The following graph displays the results of the lowest zero (β_1) on a logrithmic scale in termsof α

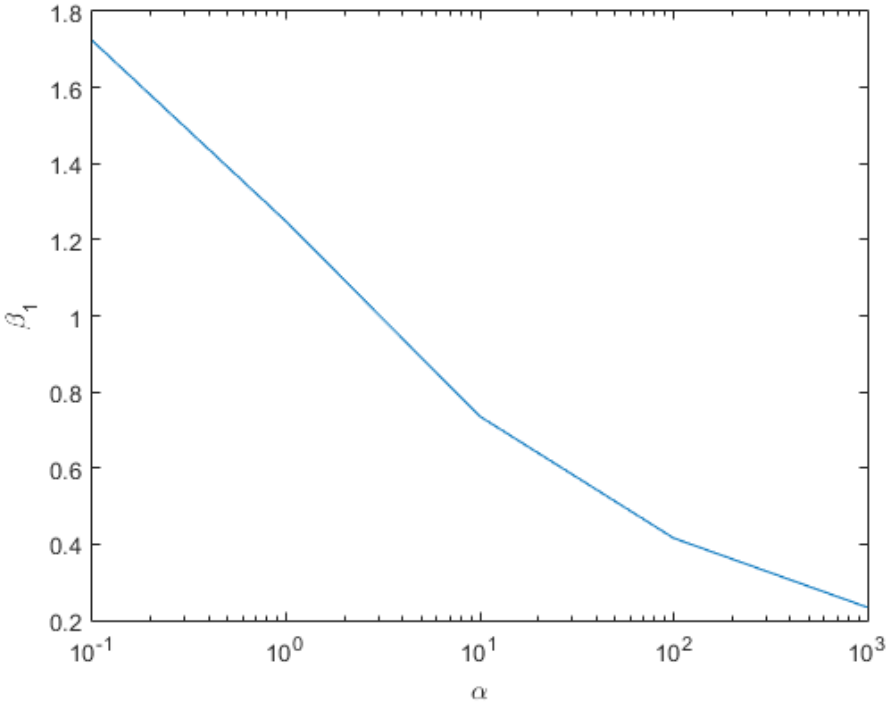

Figure 6: Graph of β_1 vs. α

Additional MATLAB code

Question 1 (h):

```
% Plot the function 1/x

% define function and domain
x = linspace(-10,10);
y = 1./x;

% plot y from [-10,10]
plot(x,y);

% add information
title('Graph of 1/x');
xlabel('-10 \leq x \leq 10');
ylabel('1/x');
```

Question 2 (a):

```
% Plot the functions sinx, 1-x^2

% define function and domain
x = linspace(0,3);
y1 = sin(x);
y2 = 1-x.^2;

% plot y from [-10,10]
plot(x,y1, x,y2);

% add information
xlabel('0\leq x \leq 3');
legend('y1 = sin(x)','y2 = 1-x^{2}','Location','northeast');
```

Question 2 (c):

```
% Plot the function sinx + x^2 - 1

% define function and domain
x = linspace(0,1);
yp = cos(x) + 2*x;
ypp = 2 - sin(x);

% plot y from [-10,10]
plot(x,yp, x,ypp);

% add information
xlabel('0\leq x \leq 1');
legend('f^{(1)}(x)', 'f^{(2)}(x)','Location','northeast');
grid on
```